Practica 2

Introducción

1. ¿Cuál es la función de la capa de aplicación?

La función de la capa de aplicación es proveer servicios de red directamente a las aplicaciones del usuario, facilitando la comunicación entre dispositivos. Define protocolos como HTTP, SMTP, FTP y DNS, que permiten la transferencia de archivos, el acceso a páginas web, el envío de correos electrónicos y la resolución de nombres de dominio.

2. Si dos procesos deben comunicarse:
   a. ¿Cómopodrían hacerlo si están en diferentes máquinas?
   b. Ysi están en la misma máquina, ¿qué alternativas existen?

a. Si los procesos están en diferentes máquinas, pueden comunicarse a través de la red usando sockets (TCP o UDP), identificándose con la dirección IP y el puerto, mediante protocolos como HTTP, FTP, DNS, etc.

b. Si los procesos están en la misma máquina, existen varias alternativas de IPC como pipes, memoria compartida, colas de mensajes, semáforos o sockets locales.

3. Explique brevemente cómo es el modelo Cliente/Servidor. Dé un ejemplo de un sistema
   Cliente/Servidor en la “vida cotidiana” y un ejemplo de un sistema informático que siga el
   modelo Cliente/Servidor. ¿Conoce algún otro modelo de comunicación?

El modelo Cliente/Servidor consiste en que un cliente solicita un servicio o recurso y un servidor lo proporciona. En la vida cotidiana, un ejemplo sería pedir comida a un delivery: el cliente realiza la solicitud y el restaurante (servidor) la satisface. En informática, un ejemplo es la navegación web, donde el navegador actúa como cliente y un servidor web (como Apache o Nginx) responde con la página solicitada. Otro modelo de comunicación es el **Peer-to-Peer (P2P)**, donde todos los nodos pueden actuar simultáneamente como clientes y servidores, como ocurre en BitTorrent.

4. Describa la funcionalidad de la entidad genérica “Agente de usuario” o “User agent”.

Un **Agente de Usuario (User Agent)** es la entidad de software que actúa como intermediario entre el usuario y los servicios de red en la capa de aplicación. Su función es enviar solicitudes a la red en nombre del usuario y presentar las respuestas recibidas. Ejemplos de user agents son los navegadores web (Chrome, Firefox) o los clientes de correo electrónico (Outlook, Thunderbird).

5. ¿Qué son y en qué se diferencian HTML y HTTP?

HTML
(HyperText Markup Language) es un lenguaje de marcado usado para estructurar y presentar la información en páginas web.
HTTP
(HyperText Transfer Protocol) es el protocolo de comunicación que permite la transferencia de información entre el cliente (navegador) y el servidor web.
En resumen: HTML define el **contenido** de la web, mientras que HTTP define el **mecanismo de transporte** de ese contenido.

6. HTTP tiene definido un formato de mensaje para los requerimientos y las respuestas.
   (Ayuda: apartado “Formato de mensaje HTTP”, Kurose).

   a. ¿Qué información de la capa de aplicación nos indica si un mensaje es de requerimiento o de respuesta para HTTP? ¿Cómo está compuesta dicha información?¿Para qué sirven las cabeceras?

   b. ¿Cuál es su formato? (Ayuda: https://developer.mozilla.org/es/docs/Web/HTTP/Headers)

   c. Suponga que desea enviar un requerimiento con la versión de HTTP 1.1 desde curl/7.74.0 a un sitio de ejemplo como www.misitio.com para obtener el recurso /index.html. En base a lo indicado, ¿qué información debería enviarse mediante encabezados? Indique cómo quedaría el requerimiento

a_ La información que distingue un requerimiento de una respuesta es la **primera línea del mensaje (start line)**: en los requerimientos contiene el método, recurso y versión de HTTP, mientras que en las respuestas contiene la versión, el código de

estado y el mensaje. Las cabeceras sirven para añadir metadatos que permiten a cliente y servidor interpretar correctamente la información (ejemplo: tipo de contenido, longitud, agente de usuario).

b_ El formato es:

- **Requerimiento:** `<Método> <Recurso> <Versión>` seguido de cabeceras y opcionalmente un cuerpo.
- **Respuesta:** `<Versión> <Código de estado> <Mensaje>` seguido de cabeceras y opcionalmente un cuerpo.

c_ Ejemplo de requerimiento HTTP/1.1 con `curl/7.74.0`:

```
GET /index.html HTTP/1.1
Host: www.misitio.com
User-Agent: curl/7.74.0
Accept: */*
```

7. Utilizando la VM, abra una terminal e investigue sobre el comando curl. Analice para qué
   sirven los siguientes parámetros (-I,-H,-X,-s).

El comando `curl` permite realizar peticiones HTTP desde la terminal.

- `I`: obtiene únicamente las cabeceras de la respuesta, sin el cuerpo.
- `H`: permite enviar o modificar cabeceras en la petición.
- `X`: especifica el método HTTP a usar (ej. GET, POST, PUT).
- `s`: activa el modo silencioso, ocultando la barra de progreso y errores.

8. Ejecute el comando curl sin ningún parámetro adicional y acceda a www.redes.unlp.edu.ar. Luego responda:
   a. ¿Cuántos requerimientos realizó y qué recibió? Pruebe redirigiendo la salida
      (>) del comando curl a un archivo con extensión html y abrirlo con un navegador.
   b. ¿Cómofuncionan los atributos href de los tags link e img en html?

c. Para visualizar la página completa con imágenes como en un navegador, ¿alcanza con realizar un único requerimiento?

d. ¿Cuántos requerimientos serían necesarios para obtener una página que tiene dos CSS, dos Javascript y tres imágenes? Diferencie cómo funcionaría
un navegador respecto al comando curl ejecutado previamente.

a_ Al ejecutar `curl www.redes.unlp.edu.ar` se realiza un único requerimiento HTTP y se recibe solamente el código HTML de la página. Si se redirige la salida a un archivo `.html` y se abre en un navegador, se verá la estructura de la página pero sin estilos ni imágenes.

b_ Los atributos `href` y `src` en HTML sirven para referenciar recursos externos: `href` se usa en enlaces y hojas de estilo, mientras que `src` se usa en imágenes y scripts. El navegador, al encontrar estos atributos, genera nuevos requerimientos para obtener dichos recursos.

c_ No alcanza con un único requerimiento para visualizar la página completa con imágenes y estilos, ya que el navegador necesita descargar además los recursos externos.

d_ Para una página con 2 CSS, 2 Javascript y 3 imágenes serían necesarios 8 requerimientos (1 para el HTML + 7 para los recursos). Un navegador hace estos requerimientos de forma automática, mientras que `curl` solo obtiene el HTML inicial.

9. Ejecute a continuación los siguientes comandos:
curl-v-s www.redes.unlp.edu.ar > /dev/null
curl-I-v-s www.redes.unlp.edu.ar

   a. ¿Qué diferencias nota entre cada uno?

   b. ¿Qué ocurre si en el primer comando se quita la redirección a /dev/null? ¿Por
qué no es necesaria en el segundo comando?

   c. ¿Cuántas cabeceras viajaron en el requerimiento? ¿Y en la respuesta

a_ La diferencia principal es que el primer comando obtiene todo el HTML de la página pero lo redirige a `/dev/null`, mostrando solo información de las cabeceras y

la conexión (`-v`), mientras que el segundo comando solo solicita las cabeceras HTTP y las muestra en pantalla.

b_ Si se quita la redirección a `/dev/null` en el primer comando, el HTML completo se imprimiría en la terminal junto con los detalles de la conexión, lo que genera mucho texto; en el segundo comando no es necesaria porque solo se descargan las cabeceras, que ocupan muy poco.

c_ En el requerimiento HTTP viajaron aproximadamente 4–6 cabeceras (Host, User-Agent, Accept, etc.), y en la respuesta viajaron aproximadamente 5–10 cabeceras (Date, Server, Content-Type, Content-Length, etc.).

10. ¿Qué indica la cabecera Date?

La cabecera `Date` indica la fecha y hora exacta en que el servidor generó la respuesta HTTP. Se utiliza para sincronización, control de caché y depuración, y siempre se expresa en formato estándar GMT.

11. En HTTP/1.0, ¿cómo sabe el cliente que ya recibió todo el objeto solicitado de manera
    completa? ¿Y en HTTP/1.1?

En HTTP/1.0, el cliente sabía que había recibido todo el objeto solicitado cuando el servidor cerraba la conexión TCP. En HTTP/1.1, gracias a las conexiones persistentes, el cliente determina el final del objeto mediante la cabecera `Content-Length` o, alternativamente, usando la codificación por bloques (chunked transfer encoding).

12. Investigue los distintos tipos de códigos de retorno de un servidor web y su significado.
    Considere que los mismos se clasifican en categorías (2XX, 3XX, 4XX, 5XX).

Los códigos de retorno HTTP indican el estado de la respuesta del servidor y se clasifican en:

- **1XX (informativos):** la petición fue recibida y se sigue procesando.
- **2XX (éxito):** la petición se completó correctamente (ej: 200 OK, 201 Created).
- **3XX (redirecciones):** se necesita otra acción, como acceder a otra URL (ej: 301, 302, 304).

- **4XX (errores del cliente):** el problema está en la petición enviada por el cliente (ej: 400, 401, 403, 404).
- **5XX (errores del servidor):** el problema está en el servidor (ej: 500, 502, 503).

13. Utilizando curl, realice un requerimiento con el método HEAD al sitio www.redes.unlp.edu.ar e indique:
    a. ¿Qué información brinda la primera línea de la respuesta?
    b. ¿Cuántos encabezados muestra la respuesta?
    c. ¿Qué servidor web está sirviendo la página?
    d. ¿El acceso a la página solicitada fue exitoso o no?¿Cuándo fue la última vez que se modificó la página?
    e. Solicite la página nuevamente con curl usando GET, pero esta vez indique que
    quiere obtenerla sólo si la misma fue modificada en una fecha posterior a la que
    efectivamente fue modificada. ¿Cómo lo hace? ¿Qué resultado obtuvo? ¿Puede
    explicar para qué sirve?

a_ Me informa el protocolo HTTP utilizado (HTTP/1.1) el codigo de estado 200 y el mensaje asociado al codigo de estado OK

```
HTTP/1.1 200 OK
```

b_ La respuesta contiene 7 encabezados:

- Date
- Server
- East-Modified
- ETag
- Accept-Ranges

- Content-Length
- Content-Type

c_ El servidor web es Apache/2.4.53 (Unix).

d_ El acceso fue exitoso (200 OK). La última modificación de la página fue el 13 de abril de 2022.

e_ Para hacer un GET condicional se usa el encabezado `If-Modified-Since`, por ejemplo:

```
curl -H "If-Modified-Since: Thu, 01 Jan 2023 00:00:00 GMT" www.redes.unlp.edu.ar
```

El resultado fue `304 Not Modified`, ya que la página no se actualizó desde 2022. Esto sirve para que los clientes y navegadores no descarguen contenido innecesariamente si no hubo cambios.

14. Utilizando curl, acceda al sitio www.redes.unlp.edu.ar/restringido/index.php y siga las instrucciones y las pistas que vaya recibiendo hasta obtener la respuesta final. Será de utilidad para resolver este ejercicio poder analizar tanto el contenido de cada página como los encabezados.

```
redes@debian:~$ curl www.redes.unlp.edu.ar/restringido/index.php
<h1>Acceso restringido</h1>

<p>Para acceder al contenido es necesario autenticarse. Para obtener los datos de acceso seguir las instrucciones detalladas en www.redes.unlp.edu.ar/obtener-usuario.php</p>
redes@debian:~$ curl www.redes.unlp.edu.ar/obtener-usuario.php
<p>Para obtener el usuario y la contraseña haga un requerimiento a esta página seteando el encabezado 'Usuario-Redes' con el valor 'obtener'</p>
redes@debian:~$ curl -H "Usuario-Redes: obtener" www.redes.unlp.edu.ar/obtener-usuario.php
<p>Bien hecho! Los datos para ingresar son:
```

Usuario: redes

Contraseña: RYC

Ahora vuelva a acceder a la página inicial con los datos anteriores.

PISTA: Investigue el uso del encabezado Authorization para el método Basi
c. El comando base64 puede ser de ayuda!
redes@debian:~$ echo -n "redes:RYC" | base64
cmVkZXM6UllD
redes@debian:~$ curl -H "Authorization: Basic cmVkZXM6UllD" www.redes.u
nlp.edu.ar/restringido/index.php
<h1>Excelente!</h1>

Para terminar el ejercicio deberás agregar en la entrega los datos que se
muestran en la siguiente página.
ACLARACIÓN: la URL de la siguiente página está contenida en esta misma
respuesta.
redes@debian:~$ curl -i -H "Authorization: Basic cmVkZXM6UllD" www.rede
s.unlp.edu.ar/restringido/index.php
HTTP/1.1 302 Found
Date: Thu, 04 Sep 2025 14:04:56 GMT
Server: Apache/2.4.53 (Unix)
X-Powered-By: PHP/7.4.28
Location: http://www.redes.unlp.edu.ar/restringido/the-end.php
Content-Length: 230
Content-Type: text/html; charset=UTF-8

<h1>Excelente!</h1>

Para terminar el ejercicio deberás agregar en la entrega los datos que se
muestran en la siguiente página.
ACLARACIÓN: la URL de la siguiente página está contenida en esta misma
respuesta.
redes@debian:~$ curl -H "Authorization: Basic cmVkZXM6UllD" www.redes.u

```
nlp.edu.ar/restringido/the-end.php
¡Felicitaciones, llegaste al final del ejercicio!

Fecha: 2025-09-04 14:07:19
Verificación: 0a6beeade13482ca98441a69d96a066d915c13436a151c9a021a2
6a59fc214a0redes@debian:~$
```

15. Utilizando la VM, realice las siguientes pruebas:
    a. Ejecute el comando ’curl www.redes.unlp.edu.ar/extras/prueba-http-1-0.txt’ y
    copie la salida completa (incluyendo los dos saltos de línea del final).
    b. Desde la consola ejecute el comando telnet www.redes.unlp.edu.ar 80 y luego pegue el contenido que tiene almacenado en el portapapeles. ¿Qué ocurre luego de hacerlo?
    c. Repita el proceso anterior, pero copiando la salida del recurso /extras/prueba-http-1-1.txt. Verifique que debería poder pegar varias veces el
    mismo contenido sin tener que ejecutar el comando telnet nuevamente.

a_

```
redes@debian:~$ curl www.redes.unlp.edu.ar/extras/prueba-http-1-0.txt
GET /http/HTTP-1.1/ HTTP/1.0
User-Agent: curl/7.38.0
Host: www.redes.unlp.edu.ar
Accept: /
```

b_

```
redes@debian:~$ telnet www.redes.unlp.edu.ar 80
Trying 172.28.0.50...
Connected to www.redes.unlp.edu.ar.
Escape character is '^]'.
```

```
GET /http/HTTP-1.1/ HTTP/1.0
User-Agent: curl/7.38.0
Host: www.redes.unlp.edu.ar
Accept: */*



HTTP/1.1 200 OK
Date: Thu, 04 Sep 2025 14:21:27 GMT
Server: Apache/2.4.53 (Unix)
Last-Modified: Wed, 13 Apr 2022 22:55:32 GMT
ETag: "760-5dc9113140100"
Accept-Ranges: bytes
Content-Length: 1888
Connection: close
Content-Type: text/html

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Protocolo HTTP: versiones</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="description" content="">
    <meta name="author" content="">

    <!-- Le styles -->
    <link href="../../bootstrap/css/bootstrap.css" rel="stylesheet">
    <link href="../../css/style.css" rel="stylesheet">
    <link href="../../bootstrap/css/bootstrap-responsive.css" rel="stylesheet">

    <!-- HTML5 shim, for IE6-8 support of HTML5 elements -->
    <!--[if lt IE 9]>
      <script src="./bootstrap/js/html5shiv.js"></script>
    <![endif]-->
  </head>
```

```
 <body>


  <div id="wrap">

  <div class="navbar navbar-inverse navbar-fixed-top">
   <div class="navbar-inner">
    <div class="container">
     <a class="brand" href="../../index.html"><i class="icon-home icon-whit
e"></i></a>
     <a class="brand" href="https://catedras.info.unlp.edu.ar" target="_blan
k">Redes y Comunicaciones</a>
     <a class="brand" href="http://www.info.unlp.edu.ar" target="_blank">F
acultad de Inform&aacute;tica</a>
     <a class="brand" href="http://www.unlp.edu.ar" target="_blank">UNLP
</a>
    </div>
   </div>
  </div>

  <div class="container">
  <h1>Ejemplo del protocolo HTTP 1.1</h1>
  <p>
   Esta p&aacute;gina se visualiza utilizando HTTP 1.1. Utilizando el captura
dor de paquetes analice cuantos flujos utiliza el navegador para visualizar la p
&aacute;gina con sus im&aacute;genes en contraposici&oacute;n con el proto
colo HTTP/1.0.
  </p>
  </p>
  <h2>Imagen de ejemplo</h2>
  <img src="13532-tuxkiller03green.png" width="800px"/>
  </div>


  </div>
```

```
    <div id="footer">
      <div class="container">
        <p class="muted credit">Redes y Comunicaciones</p>
      </div>
    </div>
  </body>
</html>
Connection closed by foreign host.
```

la conexión se cierra inmediatamente después de que el servidor envía la respuesta.

Esto es el comportamiento por defecto del protocolo **HTTP/1.0**, que establece una conexión para cada solicitud. El servidor te envía el contenido y luego corta la comunicación. Por eso, si intentas enviar otra petición, verás el mensaje `Connection closed by foreign host`.

c_ Como se puede ver en el ejemplo de abajo utilizando el protocolo HTTP 1.1 puedo realizar multiples requests sin que se cierre la conexion con el servidor.

```
redes@debian:~$ telnet www.redes.unlp.edu.ar 80
Trying 172.28.0.50...
Connected to www.redes.unlp.edu.ar.
Escape character is '^]'.
GET /extras/prueba-http-1-1.txt HTTP/1.1
Host: www.redes.unlp.edu.ar

HTTP/1.1 200 OK
Date: Thu, 04 Sep 2025 14:29:55 GMT
Server: Apache/2.4.53 (Unix)
Last-Modified: Wed, 13 Apr 2022 22:55:32 GMT
ETag: "60-5dc9113140100"
Accept-Ranges: bytes
Content-Length: 96
Content-Type: text/plain

GET /http/HTTP-1.1/ HTTP/1.1
```

```
User-Agent: curl/7.38.0
Host: www.redes.unlp.edu.ar
Accept: /



GET /extras/prueba-http-1-1.txt HTTP/1.1
Host: www.redes.unlp.edu.ar

HTTP/1.1 200 OK
Date: Thu, 04 Sep 2025 14:30:03 GMT
Server: Apache/2.4.53 (Unix)
Last-Modified: Wed, 13 Apr 2022 22:55:32 GMT
ETag: "60-5dc9113140100"
Accept-Ranges: bytes
Content-Length: 96
Content-Type: text/plain

GET /http/HTTP-1.1/ HTTP/1.1
User-Agent: curl/7.38.0
Host: www.redes.unlp.edu.ar
Accept: /
```

16. En base a lo obtenido en el ejercicio anterior, responda:
    a. ¿Qué está haciendo al ejecutar el comando telnet?
    b. ¿Qué método HTTP utilizó? ¿Qué recurso solicitó?
    c. ¿Qué diferencias notó entre los dos casos? ¿Puede explicar por qué?
    d. ¿Cuál de los dos casos le parece más eficiente? Piense en el ejercicio donde
    analizó la cantidad de requerimientos necesarios para obtener una página con estilos, javascripts e imágenes. El caso elegido, ¿puede traer asociado algún problema?

a_ Al ejecutar el comando `telnet`, estás estableciendo una **conexión TCP** de forma manual con el servidor en el puerto 80 (el puerto por defecto para el protocolo HTTP). A diferencia de `cURL` que automatiza todo el proceso, `telnet` te permite interactuar directamente con el protocolo, enviando tus propios comandos y recibiendo las respuestas del servidor sin un cliente intermedio.

b_ El método HTTP que utilice fue `GET`. Solicite el recurso `/extras/prueba-http-1-0.txt` en el primer caso (punto b del ejercicio anterior) y `/extras/prueba-http-1-1.txt` en el segundo caso (punto c del ejercicio anterior).

c_ La principal diferencia entre los dos casos fue que en la prueba con **HTTP/1.0**, la conexión se **cerró automáticamente** después de recibir la respuesta del servidor. En cambio, en la prueba con **HTTP/1.1**, la conexión **permaneció abierta**, lo que me permitió enviar una segunda petición sin tener que reconectar.

Esta diferencia se debe al manejo de la conexión de cada protocolo:

- **HTTP/1.0** está diseñado para ser de **conexión corta**. El servidor asume que una vez que envía la respuesta, el cliente no tiene más peticiones, por lo que cierra la conexión TCP para liberar recursos.
- **HTTP/1.1** introduce las **conexiones persistentes** (o `keep-alive`). El servidor mantiene la conexión activa después de enviar la respuesta, asumiendo que el cliente podría necesitar más recursos. Esto elimina la sobrecarga de tener que establecer una nueva conexión TCP por cada elemento de una página web.

d_ El caso más eficiente es **HTTP/1.1**. Para obtener una página web con múltiples elementos (como estilos, scripts e imágenes), HTTP/1.0 necesitaría establecer una nueva conexión TCP para cada uno de ellos, lo que consume tiempo y recursos. HTTP/1.1, al mantener la conexión persistente, puede solicitar todos los elementos a través de la misma conexión, reduciendo significativamente la latencia y la carga del servidor. Sin embargo, el uso de conexiones persistentes puede traer un problema: si la conexión no se cierra, el servidor puede agotar los recursos de sus conexiones activas si muchos clientes las mantienen abiertas sin necesidad. Este problema se soluciona con temporizadores que cierran la conexión si no se reciben nuevas peticiones después de un cierto período de tiempo.

17. En el siguiente ejercicio veremos la diferencia entre los métodos POST y GET. Para ello,

será necesario utilizar la VM y la herramienta Wireshark. Antes de iniciar considere:
■ Capture los paquetes utilizando la interfaz con IP 172.28.0.1. (Menú “Capture
→Options”. Luego seleccione la interfaz correspondiente y presione Start).
■ Para que el analizador de red sólo nos muestre los mensajes del protocolo http
introduciremos la cadena ‘http’ (sin las comillas) en la ventana de especificación de
filtros de visualización (display-filter). Si no hiciéramos esto veríamos todo el tráfico
que es capaz de capturar nuestra placa de red. De los paquetes que son capturados, aquel que esté seleccionado será mostrado en forma detallada en la sección que está justo debajo. Como sólo estamos interesados en http ocultaremos
toda la información que no es relevante para esta práctica (Información de trama,
Ethernet, IP y TCP). Desplegar la información correspondiente al protocolo HTTP
bajo la leyenda “Hypertext Transfer Protocol”.
■ Para borrar la cache del navegador, deberá ir al menú “Herramientas->Borrar
historial reciente”. Alternativamente puede utilizar Ctrl+F5 en el navegador para
forzar la petición HTTP evitando el uso de caché del navegador.
■ En caso de querer ver de forma simplificada el contenido de una comunicación http,
utilice el botón derecho sobre un paquete HTTP perteneciente al flujo capturado y
seleccione la opción Follow TCP Stream.

a. a. Abra un navegador e ingrese a la URL: www.redes.unlp.edu.ar e ingrese al
link en la sección “Capa de Aplicación” llamado “Métodos HTTP”. En la página mostrada se visualizan dos nuevos links llamados: Método GET y Método POST. Ambos muestran un formulario como el siguiente:

b. Analice el código HTML
c. Utilizando el analizador de paquetes Wireshark capture los paquetes enviados y recibidos al presionar el botón Enviar.
d. ¿Qué diferencias detectó en los mensajes enviados por el cliente?
e. ¿Observó alguna diferencia en el browser si se utiliza un mensaje u otro?

d_

- En GET, los parámetros viajan en la URL (en la request line).
- En POST, los parámetros viajan en el cuerpo del mensaje, separados del encabezado.

e_

- En GET, los datos aparecen en la barra de direcciones (ej: `...?nombre=Fabio`).
- En POST, los datos no aparecen en la URL, sólo se envían en el cuerpo → más seguro para contraseñas o datos sensibles.
- En cuanto al resultado de la página, normalmente no cambia lo que se muestra (el servidor puede procesar igual), pero sí cambia la forma en que se transmiten los datos.

18. Investigue cuál es el principal uso que se le da a las cabeceras Set-Cookie y Cookie en
HTTP y qué relación tienen con el funcionamiento del protocolo HTTP.

La cabecera `Set-Cookie` es enviada por el servidor para almacenar información en el navegador del cliente, mientras que la cabecera `Cookie` es enviada por el cliente en solicitudes posteriores para que el servidor pueda reconocerlo. Esto permite mantener el estado y la sesión en HTTP, que es un protocolo sin estado.

19. ¿Cuál es la diferencia entre un protocolo binario y uno basado en texto? ¿De qué tipo
de protocolo se trata HTTP/1.0, HTTP/1.1 y HTTP/2?

Un protocolo basado en texto transmite la información en formato legible, como ASCII, lo que facilita depuración pero es menos eficiente. Un protocolo binario transmite la información en formato binario, más eficiente pero difícil de leer. HTTP/1.0 y HTTP/1.1 son protocolos basados en texto, mientras que HTTP/2 es un protocolo binario.

20. Responder las siguientes preguntas:

    a. ¿Qué función cumple la cabecera Host en HTTP 1.1? ¿Existía en HTTP 1.0? ¿Qué sucede en HTTP/2? (Ayuda: https://undertow.io/blog/2015/04/27/An-in-depth-overview-of-HTTP2.html para HTTP/2)

    b. En HTTP/1.1, ¿es correcto el siguiente requerimiento?
    GET /index.php HTTP/1.1
    User-Agent: curl/7.54.0

    c. ¿Cómo quedaría en HTTP/2 el siguiente pedido realizado en HTTP/1.1 si se está usando https?
    GET /index.php HTTP/1.1
    Host: www.info.unlp.edu.ar

a_ La cabecera `Host` en HTTP/1.1 indica el dominio del servidor al que se solicita el recurso; en HTTP/1.0 no existía; en HTTP/2 se reemplaza por el pseudo-encabezado `:authority`.

b_ El requerimiento dado sin `Host` **no es correcto** en HTTP/1.1. La forma correcta incluye:

```
Host: www.info.unlp.edu.ar
```

c_ En HTTP/2 sobre HTTPS, el mismo pedido se representa con pseudo-encabezados binarios:

```
:method = GET
:scheme = https
:authority = www.info.unlp.edu.ar
:path = /index.php
```